Data Compression
Lecture Notes

Dr. Faisal Aslam

Lecture 1 Introduction to Data Compression

1.1 Learning Objectives

By the end of this lecture, students will be able to:

- Define data compression and explain its practical importance with real-world examples
- Differentiate between lossless and lossy compression with concrete applications
- Calculate and interpret basic compression metrics (compression ratio, bit-rate, savings)
- Explain the concepts of information, redundancy, and entropy with computational examples
- Identify major application domains and their specific compression requirements
- Understand the fundamental limits of compression from information theory

1.2 Introduction and Motivation: Why Compress Data?

Data compression is the process of encoding information using fewer bits than the original representation. Every day, we encounter compression without realizing it: from streaming videos to sending emails, from saving photos to downloading software updates.

Definition

Data Compression: The process of reducing the number of bits needed to represent information, while either:

- **Lossless**: Preserving all original information exactly
- **Lossy**: Accepting some controlled loss of information for higher compression

1.2.1 Real-World Motivation: A Concrete Example

Consider a typical smartphone photo: 12 megapixels, 24-bit color (8 bits per RGB channel). Uncompressed size:

$$12,000,000 \text{ pixels} \times 24 \text{ bits/pixel} = 288,000,000 \text{ bits} = 36 \text{ MB}$$

But your phone stores it as a 3 MB JPEG file. That's a 12:1 compression ratio! Without compression:

- Your 128 GB phone could store only 3,500 photos instead of 40,000
- Uploading to social media would take 12 times longer
- Cloud storage costs would be 12 times higher

1.2.2 The Economics of Compression

Example

Cloud Storage Example: A major cloud provider charges \$0.023 per GB per month. For 1 PB (petabyte = 1000 TB) of data:

- Uncompressed: 1 PB = 1,000,000 GB gives \$23,000/month
- With 4:1 compression: 250,000 GB gives \$5,750/month
- Annual savings: (\$23,000 - \$5,750) $\times$ 12 = \$207,000/year

This doesn't even consider bandwidth costs, which are typically charged per GB transferred!

1.3 Lossless vs. Lossy Compression: A Detailed Comparison

1.3.1 Lossless Compression: Perfect Reconstruction

How it works: Exploits statistical redundancy and patterns without losing information.
Key techniques:

1. **Entropy coding**: Assign shorter codes to frequent symbols (Huffman, Arithmetic)
2. **Dictionary methods**: Replace repeated patterns with references (LZ77, LZ78)
3. **Predictive coding**: Encode differences from predictions rather than raw values

Example

Text Compression Example: The word "compression" appears 100 times in a document.

- Uncompressed: "compression" = 11 characters × 8 bits = 88 bits × 100 = 8,800 bits
- Compressed: Assign code "01" (2 bits) for "compression" → 2 bits × 100 = 200 bits
- Plus dictionary entry: "compression" = 88 bits (stored once)
- Total: 200 + 88 = 288 bits vs 8,800 bits → 30:1 compression!

This is essentially how LZW (used in GIF, ZIP) works.

1.3.2 Lossy Compression: Intelligent Approximation

How it works: Removes information that is:

- Imperceptible to humans (psychovisual/psychoacoustic models)
- Less important for the intended use
- Redundant beyond a certain quality threshold

Example

JPEG Image Compression - Step by Step:

1. **Color Space Conversion**: RGB to YCbCr (separates luminance from color)
2. **Chrominance Downsampling**: Reduce color resolution (4:2:0) - humans are less sensitive to color details
3. **Discrete Cosine Transform (DCT)**: Convert 8×8 pixel blocks to frequency domain
4. **Quantization**: Divide frequency coefficients by quantization matrix - small high-frequency coefficients become zero
5. **Entropy Coding**: Huffman code the results

   Result: Typical 10:1 to 20:1 compression with minimal visible artifacts

1.3.3 When to Use Which? Decision Factors

The choice between lossless and lossy compression depends on the acceptable level of information loss, the nature of the data, and system constraints such as speed and storage. Lossless compression is required whenever exact reconstruction is mandatory, whereas lossy compression is preferred when human perception can tolerate approximations in exchange for significantly higher compression ratios. The following table summarizes the key decision factors commonly encountered in practice.

Factor	**Choose Lossless When**	**Choose Lossy When**
Fidelity Requirement	Exact reconstruction is critical (code, financial data, legal documents)	Some quality loss is acceptable (media streaming, web images)
Data Type	Discrete data with exact values (text, databases, executables)	Continuous data with perceptual limits (images, audio, video)
Compression Ratio Needed	Moderate ratios suffice (2:1 to 10:1)	High ratios needed (10:1 to 200:1+)
Processing Requirements	Fast decompression needed, encode speed less critical	Real-time encoding/decoding needed (streaming, videoconferencing)
Regulatory Constraints	Legal/medical requirements mandate exact copies	No regulatory constraints on quality

Table 1: Decision Factors for Lossless vs. Lossy Compression

1.4 Performance Metrics: Beyond Simple Ratios

1.4.1 Compression Ratio and Savings

$$\text{Compression Ratio} = \frac{\text{Original Size}}{\text{Compressed Size}}$$
$$\text{Savings} = \left(1 - \frac{\text{Compressed Size}}{\text{Original Size}}\right) \times 100\%$$

Example

Comparing Different Compression Scenarios:

Scenario	Original	Compressed	Ratio	Savings
Text document (ZIP)	1.5 MB	450 KB	3.33:1	70%
CD Audio (FLAC lossless)	700 MB	350 MB	2:1	50%
Same Audio (MP3 128kbps)	700 MB	112 MB	6.25:1	84%
4K Video (H.265)	100 GB	2 GB	50:1	98%
DNA sequence (specialized)	3 GB	300 MB	10:1	90%

1.4.2 Bit-rate: The Quality Control Knob

For lossy compression, bit-rate determines quality:

$$\text{Bit-rate} = \frac{\text{Compressed Size in bits}}{\text{Duration (seconds)}} \quad \text{or} \quad \frac{\text{Compressed Size in bits}}{\text{Number of samples}}$$

Example

Audio Quality at Different Bit-rates:

- **32 kbps**: Telephone quality, speech only
- **96 kbps**: FM radio quality
- **128 kbps**: "Good enough" for most listeners
- **192 kbps**: Near CD quality for most people
- **320 kbps**: Essentially transparent (FLAC: 900 kbps)

  Storage impact: A 60-minute album:

  - At 128 kbps: 60 MB
  - At 320 kbps: 144 MB
  - FLAC lossless: 400 MB
  - Uncompressed CD: 700 MB

1.4.3 Time and Space Trade-offs

Compression involves multiple competing objectives:

$$\text{Space–Time Trade-off} = \frac{\text{Compression Ratio}}{\text{Encoding Time} \times \text{Decoding Time}}$$

While higher compression ratios reduce storage and bandwidth, they often come at the cost of increased computational complexity and latency. In real-time and large-scale streaming systems (e.g., Netflix, YouTube, Facebook), **encoding and especially decoding speed are often more critical than optimal compression ratios**. Streaming workloads require fast, low-latency decoding on a wide range of devices, from mobile phones to smart TVs, where CPU, memory, and power budgets are limited.

As a result, practical streaming systems favor compressors that achieve a *good enough* compression ratio while providing high throughput, low memory usage, and predictable latency, even if better compression is theoretically possible.

Example

Real-world Compressor Comparison:

Algorithm	Ratio (text)	Encode Speed	Decode Speed	Memory
gzip (-6)	3.2:1	100 MB/s	400 MB/s	10 MB
bzip2 (-6)	3.8:1	20 MB/s	50 MB/s	50 MB
LZ4	2.5:1	500 MB/s	2000 MB/s	1 MB
Zstd (-3)	3.0:1	300 MB/s	800 MB/s	5 MB
xz (-6)	4.2:1	10 MB/s	80 MB/s	100 MB

Approximate performance on typical text data (higher is better)

1.5 Information and Redundancy: The Core Concepts

1.5.1 Information: Quantifying Surprise

In everyday language, information is often confused with the mere presence of data. In information theory, however, information measures how much an observation *reduces uncertainty*. If an outcome is fully predictable, observing it provides little or no new information.

Claude Shannon's key insight was that information is inversely related to probability: unlikely events are more informative because they are more surprising, while highly likely events convey little new knowledge.

Definition

Information Content of an event with probability p is defined as:

$$I(p) = -\log_2 p \quad \text{bits}$$

This definition captures an important intuition: as an event becomes more predictable ($p \to 1$), its information content approaches zero.

Example

Predictability vs. Information:

- In a city where it rains every day, the statement "It rained today" conveys almost no information because it was already expected.
- A file that contains only the bit '1' provides very little information, since after seeing a few bits, the rest of the file can be predicted with certainty.
- A coin that always lands heads produces outcomes, but no information, because there is no uncertainty to resolve.

Key idea: Perfect predictability implies zero information gain.

Example

Daily Weather Forecast — Information Content:

- Sunny in Phoenix (probability 0.9): $I = -\log_2 0.9 \approx 0.15$ bits
- Snow in Phoenix (probability 0.001): $I = -\log_2 0.001 \approx 9.97$ bits
- Rain in Seattle (probability 0.3): $I = -\log_2 0.3 \approx 1.74$ bits

Interpretation: Rare events carry more information because they reduce uncertainty the most. Snow in Phoenix reveals far more about the weather system than another sunny day.

In summary, information is not about how much data is observed, but about how much uncertainty is removed. This distinction between information and predictability forms the foundation of redundancy, entropy, and data compression.

1.5.2 Redundancy: The Enemy of Information and the Friend of Compression

Redundancy refers to predictable or repeated structure in data. From the perspective of information theory, redundancy is the *enemy of information* because it does not reduce uncertainty. However, from the perspective of data compression, redundancy is a *valuable resource*: it is precisely what allows data to be represented using fewer bits.

Compression algorithms work by identifying, modeling, and removing redundancy while preserving the underlying information (in lossless compression) or perceptually important information (in lossy compression).

Redundancy appears in several common forms:

1. **Spatial Redundancy**: Neighboring data values are highly correlated.

Example

In a photograph of a clear blue sky, most neighboring pixels have nearly identical color values.

- **Naive**: Store the RGB value of each pixel independently.
- **Smarter**: Encode repeated pixel values using run-length encoding.
- **Even smarter**: Predict each pixel from its neighbors and encode only the small prediction error.

2. **Statistical Redundancy**: Some symbols occur far more frequently than others.

Example

English letter frequencies:

Letter	Frequency	Letter	Frequency
E	12.7%	Z	0.07%
T	9.1%	Q	0.10%
A	8.2%	J	0.15%

- **Inefficient**: Fixed-length coding (5 bits per letter).
- **Efficient**: Variable-length coding (e.g., Huffman coding), where frequent letters get shorter codes.

This reduces the average bits per letter from 5 to approximately 4.1.

3. **Knowledge Redundancy**: Information already known to both encoder and decoder.

Example

Medical Imaging: Both the encoder and decoder know the image represents a chest X-ray.

- The general structure of lungs and bones does not need to be encoded explicitly.
- Anatomical models can be used to predict expected structures.
- Bits can be concentrated on unexpected or diagnostically important regions.

4. **Perceptual Redundancy**: Information that humans cannot perceive.

Example

Audio Compression (MP3):

- **Frequency masking**: Loud sounds mask nearby frequencies.
- **Temporal masking**: Loud sounds mask quieter sounds before or after them.
- **Result**: A large fraction of the audio data can be discarded without perceptible loss in quality.

1.5.3 What is Entropy? Different Perspectives

The term "entropy" appears in multiple fields (thermodynamics, information theory, statistics) with related but distinct meanings. In information theory, we primarily discuss **Shannon Entropy**, named after Claude Shannon who founded the field in 1948. While there are other entropy measures (like Kolmogorov-Sinai, Rényi, and Tsallis entropies in various contexts), Shannon entropy is the foundational concept for data compression.

Definition

Shannon Entropy of a discrete random variable X with possible values $\{x_1, x_2, \ldots, x_n\}$ having probabilities $\{p_1, p_2, \ldots, p_n\}$:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i \quad \text{bits}$$

Two Complementary Interpretations:

1. **Average Information Content**: When a symbol with probability p_i occurs, it conveys $-\log_2 p_i$ bits of information (rare events tell us more). Entropy is the *expected value* or average of this information content across all symbols.

2. **Uncertainty or Surprise**: Entropy measures how uncertain we are about the next symbol before observing it. Higher entropy means more unpredictability.

These interpretations are two sides of the same coin: *The average information gained equals the uncertainty removed by observation.*

1.5.4 Calculating Entropy: Step by Step

Let's examine both interpretations through detailed calculations:

Example

Binary Source Example - Detailed Calculation:

Consider a biased coin: P(Heads) = 0.8, P(Tails) = 0.2

Step 1: **Calculate individual information content**:

$$I_H = -\log_2(0.8) \approx 0.3219 \text{ bits}$$
$$I_T = -\log_2(0.2) \approx 2.3219 \text{ bits}$$

Interpretation: Tails (rarer event) carries more information.

Step 2: **Calculate entropy as expected value**:

$$H = 0.8 \times 0.3219 + 0.2 \times 2.3219 = 0.7219 \text{ bits}$$

Step 3: **Verify using direct formula**:

$$H = -[0.8 \log_2(0.8) + 0.2 \log_2(0.2)] \approx 0.7219 \text{ bits}$$

Key Insights:

- **Average information**: Each flip gives 0.72 bits of information on average
- **Uncertainty**: We're 72% as uncertain as with a fair coin
- **Extreme cases**:
  - Fair coin (P=0.5): $H = 1.0$ bit (maximum uncertainty/information)
  - Always heads (P=1.0): $H = 0$ bits (no uncertainty, no information)
  - 90% heads: $H \approx 0.469$ bits (less uncertainty than 80% case)

Mathematical Properties of Entropy:

- **Non-negativity**: $H(X) \geq 0$, with equality only when one outcome is certain
- **Maximum value**: For n symbols, maximum entropy is $\log_2 n$, achieved when all probabilities are equal ($p_i = 1/n$)
- **Concavity**: Entropy is a concave function of probabilities

1.5.5 Entropy of English Text: A Practical Case Study

Example

Calculating English Letter Entropy:
Based on letter frequencies in typical English text:

Letter	Probability (p_i)	$-\log_2 \mathbf{p_i}$	Contribution ($p_i \times -\log_2 p_i$)
E	0.127	2.98	0.378
T	0.091	3.46	0.315
A	0.082	3.61	0.296
O	0.075	3.74	0.281
I	0.070	3.84	0.269
N	0.067	3.90	0.261
S	0.063	3.99	0.251
H	0.061	4.04	0.246
R	0.060	4.06	0.244
D	0.043	4.54	0.195
⋮	⋮	⋮	⋮
Z	0.0007	10.48	0.007
Total	1.0		**4.18 bits**

Layered Interpretation:

- **First-order entropy (letters independent)**: 4.18 bits/letter
- **Why not 5 bits?** Because letters are not equally likely
- **Actual uncertainty is lower**: Letters have dependencies (Q is usually followed by U)
- **Comparison with encoding schemes**:

Encoding Method	Bits/Letter	Efficiency
Naive (5 bits for 26 letters)	5.00	83.6%
Huffman (letter-based)	4.30	97.2%
Using digram frequencies	3.90	107.2%*
Using word frequencies	2.30	181.7%*
Optimal with full context	1.50	278.7%*

*Percentages ¿100% show compression better than first-order entropy by exploiting dependencies

1.5.6 Beyond First-Order Entropy: The Full Picture

Higher-Order Entropy accounts for dependencies between symbols:

- **Zero-order entropy**: $H_0 = \log_2 n$ (equal probabilities)
- **First-order entropy**: $H_1 = -\sum p_i \log_2 p_i$ (letter frequencies only)
- **Second-order entropy**: $H_2 = -\sum p(i, j) \log_2 p(i|j)$ (letter pairs)
- **N-th order entropy**: $H_N = -\sum p(\text{block}) \log_2 p(\text{last}|\text{previous})$

The Entropy Rate is the limit as $N \to \infty$:

$$H_\infty = \lim_{N\to\infty} H_N$$

For English, $H_\infty \approx 1.0 - 1.5$ bits/letter, much lower than first-order entropy!

1.5.7 The Entropy Theorem: Why It Matters

Important

Shannon's Source Coding Theorem (Formal Statement):
Given a discrete memoryless source with entropy H, for any $\epsilon > 0$:

1. **Converse**: No code can have average length $L < H$ without losing information
2. **Achievability**: There exists a code with average length $L < H + \epsilon$

Implications for Compression:

- **Fundamental limit**: H is the absolute lower bound for lossless compression
- **Redundancy**: Difference between actual code length and H is wasted space
- **Optimality**: Good compressors approach H from above

Practical Example - English Text Compression:

- **Impossible**: Average $<$ 1.5 bits/letter (entropy rate limit)
- **Wasteful**: 8 bits/letter (ASCII - 533% of optimal)
- **Good**: 2.5 bits/letter (modern compressors - 167% of optimal)
- **Theoretical limit**: 1.5 bits/letter (100% efficiency)

Why We Can't Reach The Limit Exactly:

- Finite block sizes in practical codes
- Computational complexity of optimal coding
- Need for integer-length codes (Huffman)
- Model inaccuracy in estimating probabilities

Takeaway Message:

- **Entropy IS both**: average information AND uncertainty
- **First-order** H gives a baseline, but real sources have lower entropy rates
- **The theorem** tells us what's possible and impossible
- **Good compression** = modeling dependencies to approach H_∞

1.6 Application Domains: Specialized Requirements

1.6.1 Text and Code Compression

- **Requirements**: Lossless, fast random access, incremental updates
- **Challenges**: Small files, need for searching within compressed data
- **Solutions**: gzip (DEFLATE), LZ4, Zstandard

> **Example**
>
> **Git Version Control**: Uses zlib (DEFLATE) and delta compression:
>
> – Stores file versions as compressed objects
>
> – Applies delta compression for similar versions (packfiles)
>
> – Exploits low *conditional entropy* between revisions
>
> – Example: Linux kernel repository: $\sim$4 GB raw, $\sim$1 GB stored

1.6.2 Multimedia Compression

- **Requirements**: High compression, perceptual quality, real-time
- **Challenges**: Massive data volumes, human perception constraints
- **Solutions**: JPEG, MP3, H.264/HEVC, AV1

> **Example**
>
> **Streaming Service Economics (Netflix/YouTube)**:
>
> – 1 hour of 4K video: Uncompressed 500 GB
>
> – H.265 compressed: 4 GB (125:1 compression)
>
> – Bandwidth cost: $0.05/GB (typical CDN pricing)
>
> – Uncompressed stream: $25/hour
>
> – Compressed stream: $0.20/hour
>
> – For 100 million hours/day: $20M/day vs $2.5B/day!

1.6.3 Scientific and Medical Data

- **Requirements**: Lossless or controlled loss, reproducibility, standards

- **Challenges**: Huge datasets, precision requirements, regulatory compliance
- **Solutions**: Specialized compressors (SZ, ZFP), format standards (DICOM)

> **Example**
>
> **Large Hadron Collider (LHC) Data**:
>
> - Generates 1 PB/second (yes, per second!)
> - Stores 50 PB/year after filtering
> - Uses specialized compression algorithms
> - Compression saves $50M/year in storage costs
> - Enables global collaboration (data distributed worldwide)

1.7 The Compression Pipeline: How Compressors Actually Work

Most compressors follow this two-stage process:

Compression Pipeline:

$$\textbf{Input Data} \xrightarrow[\text{Analyzes patterns}]{} \textbf{Modeling Stage} \xrightarrow[\text{Builds probability model}]{}$$

$$\xrightarrow{} \textbf{Coding Stage} \xrightarrow[\text{Using entropy coding}]{}$$

Converts to bits

$$\xrightarrow{} \textbf{Compressed Data}$$

Two-Stage Compression Pipeline:

- **Modeling Stage**: Analyzes data patterns and builds probability model
- **Coding Stage**: Converts symbols to bits using entropy coding (Huffman, Arithmetic, ANS)

1.7.1 Modeling Strategies in Practice

Example

Huffman Coding Example - Complete Process:

1. **Modeling**: Count symbol frequencies in "ABRACADABRA"

Symbol	Frequency	Probability
A	5	$5/11 \approx 0.455$
B	2	$2/11 \approx 0.182$
R	2	$2/11 \approx 0.182$
C	1	$1/11 \approx 0.091$
D	1	$1/11 \approx 0.091$

2. **Coding**: Build Huffman tree (simplified):
   - Combine lowest frequencies: C(1) + D(1) = CD(2)
   - Continue combining: CD(2) + B(2) = CDB(4)
   - Combine: CDB(4) + R(2) = CDBR(6)
   - Final: CDBR(6) + A(5) = Root(11)

3. **Code assignment**:

Symbol	Code	Length
A	0	1 bit
R	10	2 bits
B	110	3 bits
C	1110	4 bits
D	1111	4 bits

4. **Compress "ABRACADABRA"**:
   - A(0) B(110) R(10) A(0) C(1110) A(0) D(1111) A(0) B(110) R(10) A(0)
   - Total bits: 1+3+2+1+4+1+4+1+3+2+1 = 23 bits
   - Original: 11 characters $\times$ 8 bits = 88 bits
   - Compression: 88 $\rightarrow$ 23 bits (3.8:1 ratio)
   - Entropy limit: $H \approx 2.04$ bits/char $\times$ 11 = 22.5 bits
   - Efficiency: 22.5/23 = 97.8% efficient!

1.8 Important Terminology and Concepts

1.8.1 Key Definitions with Examples

- **Symbol**: The basic unit being compressed

Example

Different domains use different symbols:

  - Text: Characters (bytes)
  - Images: Pixels (RGB triples)
  - Audio: Samples (16-bit integers)
  - Video: Macroblocks (16×16 pixel regions)

- **Alphabet**: Set of all possible symbols

Example

  - English text: 256 possible bytes (ASCII/UTF-8)
  - Binary data: 256 possible byte values
  - DNA sequences: 4 symbols {A, C, G, T}
  - Black-white image: 2 symbols {0=black, 1=white}

- **Prefix Code**: Crucial for instant decoding

Example

Why prefix codes matter:

  - Good: A=0, B=10, C=110, D=111
  - ”010110” decodes unambiguously: A(0) B(10) C(110)
  - Bad: A=0, B=1, C=01 (not prefix-free)
  - ”01” could be AB or C - ambiguous!

1.8.2 The Fundamental Insight

Important

The Core Principle of Compression:

- **Random data cannot be compressed**: Maximum entropy = no redundancy
- **Real-world data is not random**: Contains patterns, structure, predictability
- **Compression finds and exploits these patterns**

  Example - Encryption vs Compression:

  - Encrypted data looks random (high entropy)
  - Compressing encrypted data gives little or no savings
  - Always compress **before** encrypting, not after!
  - Rule: Encrypt $\rightarrow$ High entropy $\rightarrow$ No compression
  - Rule: Compress $\rightarrow$ Lower entropy $\rightarrow$ Then encrypt

Homework Assignment 1: Fundamentals of Data Compression

Assignment Details

Due:	One week after the class
Total Points:	100 points
Objective:	This assignment reinforces the fundamental concepts introduced in Lecture 1: compression ratios, entropy, redundancy, and the practical trade-offs in data compression.

Part 1: Compression Metrics and Real-World Calculations (30 points)

1. **Video Storage Calculations (15 points)**

   A streaming service stores 4K video with the following specifications:

   - Resolution: 3840×2160 pixels

- Color depth: 24 bits per pixel
- Frame rate: 24 frames per second
- Average video duration: 2 hours

Calculate:

(a) The uncompressed size of one frame in megabytes (MB)

(b) The uncompressed size of the entire 2-hour video in terabytes (TB)

(c) The compressed size if using a lossy codec with a 50:1 compression ratio

(d) The bandwidth required to stream the compressed video in real-time (in Mbps)

Show all calculations step by step.

2. **Economic Impact Analysis (15 points)**

A cloud storage company charges \$0.023 per GB per month. A client needs to store:

- 50 TB of medical images (lossless compression, average ratio 3:1)
- 200 TB of surveillance video (lossy compression, average ratio 20:1)
- 10 TB of legal documents (lossless compression, average ratio 2.5:1)

Calculate:

(a) The monthly and annual storage costs WITHOUT compression

(b) The monthly and annual storage costs WITH compression

(c) The annual savings due to compression

(d) Assuming bandwidth costs of \$0.05 per GB transferred, how much would compression save if 10% of this data is downloaded each month?

Part 2: Entropy and Information Theory (30 points)

3. **Basic Entropy Calculations (10 points)**

Calculate the Shannon entropy (in bits) for the following sources:

(a) A fair six-sided die

(b) Weather in a desert: Sunny (probability 0.85), Cloudy (0.10), Rainy (0.05)

(c) A binary source where 0 appears with probability 0.99 and 1 with probability 0.01

Show the entropy formula with values substituted for each case.

4. **Interpretation and Analysis (10 points)**

   Based on your calculations from question 3:

   (a) Which source has the highest entropy? Why?

   (b) Which source is most compressible? Explain using the concept of redundancy.

   (c) If you observed "Rainy" in the desert weather example, how many bits of information would this convey? Show your calculation.

   (d) What does an entropy of 0 bits mean practically?

5. **English Text Entropy Analysis (10 points)**

   Consider the first-order letter frequencies in English:

   - E: 12.7%, T: 9.1%, A: 8.2%, O: 7.5%, I: 7.0%, N: 6.7%, S: 6.3%, H: 6.1%
   - The remaining 18 letters share the remaining 42.4% (you may assume equal distribution for simplicity)

   (a) Calculate the first-order entropy of English text using these frequencies

   (b) Compare this to the 5 bits needed for a naive fixed-length encoding of 26 letters

   (c) Explain why actual compression algorithms can achieve better than 4.18 bits/letter in practice

Part 3: Lossless vs. Lossy Compression Analysis (25 points)

6. **Scenario-Based Decision Making (15 points)**

   For each scenario below, recommend either lossless or lossy compression and justify your answer with at least two reasons from the lecture:

   (a) Archiving a software source code repository

   (b) Streaming music to mobile devices over cellular data

   (c) Storing MRI scans in a hospital database

   (d) Video conferencing with limited bandwidth

   (e) Backing up a financial transactions database

7. **Compression Ratio Comparison (10 points)**

   The following compression ratios are achieved on different file types:

   - Text document (.txt): 3.5:1
   - CD Audio (.wav to .flac): 2.2:1
   - Same audio (.wav to .mp3 at 128 kbps): 6.8:1

- 4K Video (uncompressed to H.265): 55:1
- Digital photo (RAW to JPEG): 8:1

(a) Calculate the percentage savings for each case

(b) Explain why video achieves much higher compression ratios than audio

(c) Why does lossless audio compression achieve relatively low ratios compared to lossy?

Part 4: Practical Investigation and Critical Thinking (15 points)

8. **File Compression Experiment (10 points)**

   Find three files on your computer: a plain text file (.txt), a Microsoft Word document (.docx), and a JPEG image (.jpg). For each:

   (a) Record the original file size

   (b) Compress it using ZIP compression (standard settings)

   (c) Record the compressed size

   (d) Calculate the compression ratio

   (e) Explain the results based on the concepts of redundancy and entropy discussed in class

   Note: You can use built-in compression tools on your operating system.

9. **Future Trends Analysis (5 points)**

   Based on the lecture's discussion of application domains:

   (a) Identify one emerging application that will create new compression challenges

   (b) What type of compression (lossless/lossy) would it likely use and why?

   (c) What specific requirements might it have (e.g., real-time, ultra-high compression, etc.)?

Submission Guidelines

- Submit a PDF document with all calculations, answers, and explanations
- Show all work for mathematical calculations
- Write explanations in complete sentences
- For Part 4, include screenshots or clear documentation of your file compression experiment

Grading Rubric

Section	Points	Criteria
Part 1	30	Correct calculations, proper units, clear steps
Part 2	30	Accurate entropy calculations, correct interpretations
Part 3	25	Reasoned justifications, application of lecture concepts
Part 4	15	Thoughtful analysis, clear documentation
Total	**100**	

Lecture 2 Shannon's Source Coding Theorem and Huffman Coding

2.1 Learning Objectives

By the end of this lecture, students will be able to:

- Formally state and prove Shannon's Source Coding Theorem for discrete memoryless sources
- Apply the Kraft-McMillan inequality to characterize uniquely decodable codes
- Construct optimal prefix codes using Huffman's algorithm
- Derive and interpret the relationship between entropy and achievable compression rates
- Compute code efficiency, redundancy, and performance bounds
- Understand practical algorithms that approach the theoretical limits

Important

Big Picture:

- **Entropy** = Theoretical limit of lossless compression
- **Kraft inequality** = Feasibility condition for prefix codes
- **Huffman coding** = Optimal prefix code construction
- **Block coding** = Approach entropy by coding symbols in blocks
- **Integer constraint** = Source of the "+1" overhead in Shannon's theorem
- **Arithmetic coding** = Removes integer constraint using fractional bits

	Method	**Purpose**	**Key Propert**
Three Key Coding Methods:	Shannon coding	Proof of achievability	$\ell_i = \lceil -\log_2 p_i$
	Huffman coding	Optimal prefix code	Minimizes exp
	Arithmetic coding	Near-optimal compression	Fractional bits

2.2 Mathematical Preliminaries and Notation

Let X be a discrete random variable taking values in alphabet $\mathcal{X} = \{x_1, x_2, \ldots, x_m\}$ with probability mass function $p(x) = \Pr(X = x)$.

Definition

Source Code: A mapping $C : \mathcal{X} \to \mathcal{D}^*$ where $\mathcal{D} = \{0, 1\}$ is the code alphabet, and $\mathcal{D}^*$ is the set of all finite binary strings. The code C assigns to each symbol x_i a codeword c_i of length $\ell_i = |c_i|$.

2.2.1 Expected Code Length

For a source with probabilities $p_1, p_2, \ldots, p_m$ and corresponding codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$, the expected code length is:

$$L(C) = \mathbb{E}[\ell(X)] = \sum_{i=1}^{m} p_i \ell_i$$

2.3 Shannon's Source Coding Theorem: Formal Statement

Definition

Shannon's Source Coding Theorem (1948):

- **Converse (Lower Bound)**: For any uniquely decodable code C for a discrete memoryless source X with entropy $H(X)$, the expected length satisfies:

$$L(C) \geq H(X)$$

- **Achievability (Upper Bound)**: There exists a uniquely decodable code C such that:

$$L(C) < H(X) + 1$$

- **Block Coding**: For the nth extension of the source, there exists a uniquely decodable code C_n such that:

$$\frac{1}{n}L(C_n) \to H(X) \quad \text{as } n \to \infty$$

2.3.1 Interpretation and Significance

- **Fundamental Limit**: $H(X)$ bits/symbol is the **asymptotic lower bound** for lossless compression
- **Achievability**: We can get arbitrarily close to this limit by coding in blocks
- **Penalty Term**: The "+1" represents overhead from integer codeword lengths
- **The Integer Constraint**: Codeword lengths must be integers, but ideal lengths $-\log_2 p_i$ are typically not integers
- **Important**: The "+1" term is **not inefficiency of Huffman**, but a fundamental limitation of integer-length codes

Example

Binary Source Analysis: Consider a binary source with $P(0) = p$, $P(1) = 1-p$:

- Entropy: $H(p) = -p \log_2 p - (1-p)\log_2(1-p)$
- For $p = 0.1$: $H(0.1) \approx 0.469$ bits/symbol
- Theorem guarantees: $0.469 \leq L < 1.469$ bits/symbol
- Simple code: 0→0, 1→1 gives $L = 1$ bit/symbol (efficiency 46.9%)
- Block coding can approach 0.469 bits/symbol

2.4 Code Classification and Properties

2.4.1 Hierarchical Classification of Codes

Hierarchy of Codes:

All Codes
↓
Non-singular Codes
↓
Uniquely Decodable Codes
↓
Prefix Codes

2.4.2 Formal Definitions

1. **Non-singular**: $C(x_i) \neq C(x_j)$ for $i \neq j$
2. **Uniquely Decodable**: Every finite concatenation of codewords can be decoded in exactly one way
3. **Prefix (Instantaneous)**: No codeword is a prefix of another

Important

Hierarchy Theorem: Every prefix code is uniquely decodable, but not vice versa. However, for every uniquely decodable code, there exists a prefix code with the same codeword lengths (McMillan's theorem).

Example

Code Classification Examples:

Code	Mapping	Singular?	Uniquely Decodable?	Prefix?
C_1	a→0, b→0, c→1	Yes	No	No
C_2	a→0, b→01, c→11	No	No	No
C_3	a→0, b→01, c→011	No	No	No
C_4	a→0, b→10, c→110	No	Yes	Yes

Table 2: Classification of different codes for alphabet $\{a, b, c\}$

Analysis of C_3: The string "011" is ambiguous: it could be parsed as "ab" (0 followed by 11) or as "c" (011). Since there exists a string with multiple valid parsings, C_3 is not uniquely decodable.

2.5 Kraft-McMillan Inequality: Mathematical Foundation

Theorem 1 (Kraft-McMillan Inequality)**.** *For any prefix code (or more generally, any uniquely decodable code) with codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ over a D-ary alphabet:*

$$\sum_{i=1}^{m} D^{-\ell_i} \leq 1$$

where D is the size of the code alphabet (2 for binary).

2.5.1 Proof Sketch for Binary Prefix Codes

1. Consider a complete binary tree of depth $L = \max_i \ell_i$
2. Each codeword of length ℓ_i occupies $2^{L-\ell_i}$ leaf positions
3. Total occupied positions: $\sum_{i=1}^{m} 2^{L-\ell_i} \leq 2^L$
4. Dividing by 2^L: $\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$

Note: The tree-based proof above applies to prefix codes; the extension to uniquely decodable codes follows from McMillan's inequality.

2.5.2 Converse: Constructing Codes from Lengths

Theorem 2 (Kraft Inequality Converse)**.** *If integers $\ell_1, \ell_2, \ldots, \ell_m$ satisfy $\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$, then there exists a binary prefix code with these lengths.*

Example

Verifying Kraft Inequality:

1. Consider lengths $\{1, 2, 3, 3\}$:

$$\sum 2^{-\ell_i} = 2^{-1} + 2^{-2} + 2^{-3} + 2^{-3} = 0.5 + 0.25 + 0.125 + 0.125 = 1$$

   A prefix code exists (e.g., 0, 10, 110, 111)

2. Consider lengths $\{1, 1, 2\}$:

$$\sum 2^{-\ell_i} = 2^{-1} + 2^{-1} + 2^{-2} = 0.5 + 0.5 + 0.25 = 1.25 > 1$$

   No prefix code exists with these lengths! This violates the Kraft inequality.

2.6 Optimal Code Lengths and Shannon Coding

2.6.1 Shannon's Length Assignment

For a source with probabilities p_i, Shannon proposed the length assignment:

$$\ell_i = \lceil -\log_2 p_i \rceil$$

where $\lceil x \rceil$ is the ceiling function.

Theorem 3. *The lengths $\ell_i = \lceil -\log_2 p_i \rceil$ satisfy the Kraft inequality.*

Proof. Since $\ell_i \geq -\log_2 p_i$, we have $-\ell_i \leq \log_2 p_i$, so:

$$2^{-\ell_i} \leq p_i \quad \Rightarrow \quad \sum_{i=1}^{m} 2^{-\ell_i} \leq \sum_{i=1}^{m} p_i = 1$$

□

Example

Shannon Coding Example: Source with probabilities $\{0.4, 0.3, 0.2, 0.1\}$

1. Compute ideal lengths: $-\log_2 p_i = \{1.32, 1.74, 2.32, 3.32\}$
2. Ceiling gives: $\ell_i = \{2, 2, 3, 4\}$
3. Check Kraft: $2^{-2}+2^{-2}+2^{-3}+2^{-4} = 0.25+0.25+0.125+0.0625 = 0.6875 \leq 1$
4. Expected length: $L = 0.4 \times 2 + 0.3 \times 2 + 0.2 \times 3 + 0.1 \times 4 = 2.4$ bits/symbol
5. Entropy: $H = 1.846$ bits/symbol
6. Efficiency: $\eta = 1.846/2.4 = 76.9\%$

2.7 Detailed Proof of Shannon's Theorem

2.7.1 Lower Bound: $L \geq H(X)$

Proof. Let p_i be symbol probabilities and ℓ_i be codeword lengths of a uniquely decodable code. From Kraft-McMillan:

$$\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$$

Define $r_i = 2^{-\ell_i} / \sum_{j=1}^{m} 2^{-\ell_j}$, so $\{r_i\}$ is a probability distribution.

Using the non-negativity of KL-divergence:

$$D(p \| r) = \sum_{i=1}^{m} p_i \log_2 \frac{p_i}{r_i} \geq 0$$

Substituting r_i:

$$\sum_{i=1}^{m} p_i \log_2 p_i - \sum_{i=1}^{m} p_i \log_2 2^{-\ell_i} + \sum_{i=1}^{m} p_i \log_2 \left(\sum_{j=1}^{m} 2^{-\ell_j} \right) \geq 0$$

Since $\sum_{j=1}^{m} 2^{-\ell_j} \leq 1$, the last term is ≤ 0, giving:

$$-H(X) + \sum_{i=1}^{m} p_i \ell_i \geq 0 \quad \Rightarrow \quad L \geq H(X)$$

$\square$

2.7.2 Upper Bound: $L < H(X) + 1$

Proof. Choose $\ell_i = \lceil -\log_2 p_i \rceil$. Then:

$$-\log_2 p_i \le \ell_i < -\log_2 p_i + 1$$

Multiply by p_i and sum over i:

$$-\sum_{i=1}^{m} p_i \log_2 p_i \le \sum_{i=1}^{m} p_i \ell_i < -\sum_{i=1}^{m} p_i \log_2 p_i + \sum_{i=1}^{m} p_i$$

$$H(X) \le L < H(X) + 1$$

□

2.7.3 The Integer Length Constraint

The "+1" term in Shannon's theorem arises from the integer constraint on codeword lengths. For a **dyadic source** where all probabilities are of the form $p_i = 2^{-k_i}$ for integers k_i, we have $-\log_2 p_i = k_i$, which are integers. In this special case, we can achieve $L = H$ exactly.

2.8 Extended Source Coding and Block Codes

2.8.1 The nth Extension of a Source

For a discrete memoryless source X, the nth extension $X^n = (X_1, X_2, \ldots, X_n)$ has:

$$H(X^n) = nH(X)$$

Applying Shannon's theorem to X^n gives a code C_n with:

$$nH(X) \le L(C_n) < nH(X) + 1$$

Thus, the average length per symbol satisfies:

$$H(X) \le \frac{L(C_n)}{n} < H(X) + \frac{1}{n}$$

Example

Block Coding Improvement: Binary source with $p(0) = 0.9$, $p(1) = 0.1$, $H = 0.469$ bits/symbol

- Single symbol coding: Best code gives $L = 1$ bit/symbol (efficiency 46.9%)
- **Block coding with** $n = 2$: Consider coding pairs of symbols:

$$\begin{aligned}
00 &: (0.9)^2 = 0.81 \\
01 &: 0.9 \times 0.1 = 0.09 \\
10 &: 0.1 \times 0.9 = 0.09 \\
11 &: (0.1)^2 = 0.01
\end{aligned}$$

- Applying Shannon coding: $\ell_i = \lceil -\log_2 p_i \rceil$ gives lengths $\{1, 4, 4, 7\}$
- Expected length: $L_2 = 0.81 \times 1 + 0.09 \times 4 + 0.09 \times 4 + 0.01 \times 7 = 1.6$ bits/block
- Per symbol: $L_2/2 = 0.8$ bits/symbol (efficiency 58.6%)
- **Note**: This Shannon coding is generally suboptimal compared to Huffman coding; we'll see better Huffman codes next.
- As $n \to \infty$: $L_n/n \to 0.469$ bits/symbol (100% efficiency)

2.9 Code Efficiency and Redundancy Analysis

2.9.1 Performance Metrics

Definition

For a code C with expected length L coding a source with entropy H:

$$\text{Efficiency: } \eta = \frac{H}{L} \times 100\% \quad \text{Redundancy: } \rho = L - H$$

2.9.2 Theoretical Bounds

From Shannon's theorem:

$$\frac{H}{H+1} \leq \eta \leq 1 \quad \text{and} \quad 0 \leq \rho < 1$$

Example

Efficiency vs. Entropy:

H (bits/symbol)	Minimum η	Maximum ρ	Interpretation
0.1	9.1%	0.9 bits	Very compressible, but +1 term dominates
1.0	50%	1.0 bit	Worst-case Shannon bound for sources with $H = 1$
2.0	66.7%	1.0 bit	+1 becomes less significant
4.0	80%	1.0 bit	High entropy, good efficiency possible
7.0	87.5%	1.0 bit	+1 overhead relatively small

Table 3: Theoretical limits on code efficiency for different entropy values

2.10 Huffman Coding: Optimal Prefix Code Construction

2.10.1 The Huffman Algorithm: Step-by-Step

Theorem 4 (Huffman, 1952). *For a given source with symbol probabilities $p_1, p_2, \ldots, p_m$, the Huffman algorithm produces a prefix code that minimizes the expected code length $L = \sum p_i \ell_i$.*

Algorithm 1 Huffman Code Construction (Complete Version)

Require: Symbols $x_1, \ldots, x_m$ with probabilities $p_1, p_2, \ldots, p_m$
Ensure: Optimal binary prefix code

```
Create a min-priority queue Q initialized with m nodes, each containing one symbol
    and its probability
while |Q| > 1 do
    a ← EXTRACT-MIN(Q) {Node with smallest probability}
    b ← EXTRACT-MIN(Q) {Node with next smallest probability}
    Create new node z with p_z = p_a + p_b
    Make a left child of z, b right child of z
    INSERT(Q, z)
end while
The remaining node in Q is the root of the Huffman tree
Traverse tree from root to leaves, assigning 0 to left edges, 1 to right edges
```

Example

Huffman Coding Example: Source with probabilities:

$$p(A) = 0.4, \quad p(B) = 0.3, \quad p(C) = 0.2, \quad p(D) = 0.1$$

1. **Step 1**: Combine C(0.2) and D(0.1) $\rightarrow$ CD(0.3)
2. **Step 2**: Combine B(0.3) and CD(0.3) $\rightarrow$ BCD(0.6)
3. **Step 3**: Combine A(0.4) and BCD(0.6) $\rightarrow$ Root(1.0)
4. **Codes**: A=0 (1 bit), B=10 (2 bits), C=110 (3 bits), D=111 (3 bits)

Huffman Tree Visualization:

Root (1.0)
0 1
A (0.4) BCD (0.6)
0 1
B (0.3) CD (0.3)
0 1
C (0.2) D (0.1)

Analysis:

- Expected length: $L = 0.4 \times 1 + 0.3 \times 2 + 0.2 \times 3 + 0.1 \times 3 = 1.9$ bits/symbol
- Entropy: $H = 1.846$ bits/symbol
- Efficiency: $\eta = 1.846/1.9 = 97.1\%$
- Shannon code (from earlier): $L = 2.4$ bits, $\eta = 76.9\%$
- **Huffman is significantly better than Shannon coding!**

2.10.2 Huffman vs. Shannon's Theorem

- **Shannon's Theorem**: Proves $H \leq L < H + 1$ is achievable
- **Shannon Coding**: Simple construction with $\ell_i = \lceil -\log_2 p_i \rceil$
- **Huffman Coding**: Optimal construction that minimizes L
- **Relationship**: $L_{\text{Huffman}} \leq L_{\text{Shannon}} \leq H + 1$
- **For dyadic sources**: Both achieve $L = H$ exactly

2.10.3 Huffman Code Properties

Theorem 5 (Huffman Code Length Bound)**.** *For a Huffman code with codeword lengths ℓ_i, each length satisfies:*

$$\ell_i \leq \lceil -\log_2 p_i \rceil$$

That is, no Huffman codeword is longer than the corresponding Shannon codeword.

- **Optimality**: Minimizes expected code length among all prefix codes
- **Uniqueness**: **Not unique in general**. If all probabilities are distinct and no ties occur during merging, the code lengths are unique up to relabeling; otherwise multiple optimal trees may exist.
- **Length bound**: $\ell_i \leq m - 1$ for m symbols
- **Two least probable**: Always have same length, differ only in last bit
- **Kraft inequality**: Huffman codes satisfy $\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$, with equality if and only if the tree is complete

2.10.4 Optimal Code Structure Lemma

Lemma 1. *In an optimal prefix code:*

1. *If $p_j > p_k$, then $\ell_j \leq \ell_k$ (more probable = shorter code)*
2. *The two least probable symbols have the same length*
3. *The two least probable symbols differ only in the last bit*

2.10.5 Optimality Proof of Huffman Coding

Proof Sketch. By induction on number of symbols m:

Base case ($m = 2$): Trivial - need exactly 1 bit per symbol.

Inductive step: Assume Huffman optimal for $m - 1$ symbols.

Let x and y be two least probable symbols. Consider reduced alphabet where x and y are merged into z with $p_z = p_x + p_y$.

1. If C' is optimal for reduced alphabet, then creating C by splitting z into x and y (appending 0 and 1) gives:

$$L(C) = L(C') + p_x + p_y$$

2. Any optimal code for original alphabet must have x and y as siblings (same parent)
3. Huffman algorithm finds such sibling pairing

4. By induction hypothesis, C' is optimal for reduced alphabet
5. Therefore C is optimal for original alphabet

□

2.10.6 Redundancy Bound

Theorem 6 (Gallager's Redundancy Bound, 1978)**.** *For a Huffman code, the redundancy is bounded by:*

$$L - H < p_{min} + 0.086$$

where p_{min} is the smallest symbol probability. Moreover:

- *This bound is tight (achievable for some distributions)*
- *For dyadic sources ($p_i = 2^{-k_i}$): $L = H$ exactly*
- *Worst-case redundancy occurs when $p_{min} \approx 0$*

Example

Redundancy Analysis:

- **Best-case**: For dyadic source with $p_i = 2^{-k_i}$, $L = H$ exactly
- **Worst-case**: When smallest probability is very small
- **Example**: For source with probabilities $\{0.999, 0.001\}$:
  - Huffman codes: 0→0, 1→1 (1 bit each)
  - $L = 1$ bit/symbol, $H \approx 0.011$ bits/symbol
  - Redundancy: $\rho \approx 0.989$ bits/symbol
  - Efficiency: $\eta \approx 1.1\%$

2.11 Extended Huffman Coding: Approaching the Entropy Limit

2.11.1 The Problem with Basic Huffman

Even optimal Huffman coding suffers from the "+1" term in Shannon's theorem:

$$L < H(X) + 1$$

For low-entropy sources, this overhead is significant:

Example

Binary source with $p(0) = 0.9$, $p(1) = 0.1$:

- Entropy: $H(0.9) = 0.469$ bits/symbol
- Basic Huffman: Symbols {0,1}, codes {0,1}, $L = 1$ bit/symbol
- Efficiency: $\eta = 46.9\%$ (poor!)
- Problem: Overhead 0.531 bits/symbol ¿ entropy itself!

2.11.2 Block Huffman Coding Solution

Code n symbols together as ”super-symbols”:

1. Consider nth extension of source: $X^n = (X_1, X_2, \ldots, X_n)$
2. Alphabet size grows to m^n sequences
3. Apply Huffman coding to these blocks

Theorem 7 (Extended Huffman Performance)**.** *For the nth extension coded with Huffman, the average length per symbol satisfies:*

$$\lim_{n\to\infty} \frac{L_n}{n} = H(X)$$

Note: While Huffman coding typically performs at least as well as Shannon coding, it lacks a universal closed-form finite-n bound like $L_n/n < H + 1/n$.

Example

Extended Huffman for Binary Source $p(0) = 0.9$**,** $p(1) = 0.1$:

$n = 2$ **blocks (4 sequences)**:

- Probabilities: $P(00) = 0.81$, $P(01) = 0.09$, $P(10) = 0.09$, $P(11) = 0.01$
- Huffman codes: 00→0, 01→10, 10→110, 11→111
- Expected length: $L_2 = 0.81 \times 1 + 0.09 \times 2 + 0.09 \times 3 + 0.01 \times 3 = 1.29$ bits/block
- Per symbol: $L_2/2 = 0.645$ bits/symbol
- Efficiency: $\eta = 0.469/0.645 = 72.7\%$ (vs 46.9% for $n = 1$)

$n = 3$ **blocks (8 sequences)**:

- Probabilities:

$$\begin{array}{llll} 000 : 0.729, & 001 : 0.081, & 010 : 0.081, & 011 : 0.009 \\ 100 : 0.081, & 101 : 0.009, & 110 : 0.009, & 111 : 0.001 \end{array}$$

- **Note**: Showing optimality for large n is complex. The following is illustrative, not proven optimal.
- One possible Huffman assignment:

$$\begin{array}{llll} 000 \to 0, & 001 \to 100, & 010 \to 101, & 011 \to 11000 \\ 100 \to 1101, & 101 \to 11001, & 110 \to 1110, & 111 \to 1111 \end{array}$$

- Expected length: $L_3 \approx 1.6$ bits/block
- Per symbol: $L_3/3 \approx 0.533$ bits/symbol
- Efficiency: $\eta = 0.469/0.533 \approx 88.0\%$

Trend: As n increases, $L_n/n \to H(X)$

2.11.3 Practical Issues with Block Huffman

- **Exponential complexity**: Alphabet size grows as m^n
- **Memory requirements**: Need to store $2m^n - 1$ nodes in Huffman tree
- **Delay**: Must wait for n symbols before encoding/decoding

- **Adaptation**: Probabilities may change over time

Important

Warning: Theoretical vs. Practical Use of Block Huffman Block Huffman coding demonstrates that we can approach the entropy limit arbitrarily closely, but it is **of theoretical interest only for large n**. The exponential growth in complexity makes it impractical for real applications with large alphabets or large block sizes.

Example

Complexity Growth for English Text ($m = 256$):

n	Block Size	# Sequences	Tree Nodes
1	1 byte	256	511
2	2 bytes	65,536	131,071
3	3 bytes	16.7 million	33.5 million
4	4 bytes	4.3 billion	8.6 billion
5	5 bytes	1.1 trillion	2.2 trillion

Table 4: Exponential growth makes large n impractical

2.12 Adaptive Huffman Coding: Real-time Solution

2.12.1 The FGK Algorithm (Faller-Gallager-Knuth)

- **Dynamic**: Updates code as symbols arrive
- **No initial model**: Learns probabilities on-the-fly
- **Sibling property**: Maintains optimality after each update
- **Complexity**: $O(m)$ worst-case per update; can be reduced with careful data structures
- **Applications**: Used in early versions of UNIX compress, modem protocols

2.12.2 Comparison: Static vs Adaptive vs Block

Method	Efficiency	Complexity	Delay
Static Huffman	High if model good	$O(m \log m)$	None
Adaptive Huffman	Medium-High	$O(m)$ per symbol	None
Block Huffman (n)	Approaches optimal	$O(m^n \log m^n)$	n symbols

Table 5: Trade-offs in Huffman coding variants

Important

Why Arithmetic Coding Beats Huffman Huffman coding is limited by the **integer constraint** on codeword lengths, leading to the "+1" overhead in Shannon's theorem. Arithmetic coding, which we'll cover next, uses **fractional bits** and can achieve average lengths arbitrarily close to $H(X)$ without block coding, removing the "+1" penalty entirely.

2.13 Practical Implications and Limitations

2.13.1 Assumptions of Shannon's Theorem

- **Discrete Memoryless Source**: Symbols independent and identically distributed
- **Known Distribution**: Probabilities p_i are known in advance
- **Arbitrary Delay**: Block coding allows infinite delay for encoding/decoding
- **No Complexity Constraints**: No limits on computational resources

2.13.2 Violations in Practice

Example

Real-world Violations:

- **Dependencies**: English text has strong correlations between letters
- **Unknown Distribution**: Must estimate probabilities from data
- **Delay Constraints**: Real-time applications limit block size
- **Complexity**: Exponential growth with block size (m^n sequences)
- **Solution Approaches**: Adaptive coding, dictionary methods (LZ), arithmetic coding

2.14 Extensions and Generalizations

2.14.1 Markov Sources

For a kth order Markov source with conditional entropy $H(X|X^k)$, the theorem extends to:

$$H(X|X^k) \leq L < H(X|X^k) + 1$$

2.14.2 Universal Coding

When the source distribution is unknown, universal codes achieve:

$$\frac{1}{n} L_n \to H(X) \quad \text{almost surely}$$

Examples: Lempel-Ziv codes, arithmetic coding with adaptive models.

2.14.3 Rate-Distortion Theory

For lossy compression with distortion D, the rate-distortion function $R(D)$ gives the minimum achievable rate:

$$R(D) = \min_{p(\hat{x}|x):\mathbb{E}[d(X,\hat{X})] \leq D} I(X; \hat{X})$$

2.15 Advanced Examples and Applications

Example

DNA Sequence Compression: Alphabet $\{A, C, G, T\}$ with typical probabilities $\{0.3, 0.2, 0.2, 0.3\}$

- Entropy: $H = 1.97$ bits/base
- Simple code: 2 bits/base (efficiency 98.5%)
- Exploiting dependencies: Adjacent bases are correlated in genomes
- Conditional entropy: $H(X_n|X_{n-1}) \approx 1.5$ bits/base
- Practical compressors achieve 1.6 bits/base

Example

Image Compression Limit: Grayscale image with 256 levels

- Naive: 8 bits/pixel
- Actual entropy from pixel correlations: Typically 1-4 bits/pixel
- PNG (lossless): 2-6 bits/pixel (uses filtering + DEFLATE = LZ77 + Huffman)
- JPEG (lossy): 0.5-2 bits/pixel with visual quality
- Theoretical limit from image statistics

Historical Notes

- **1948**: Claude Shannon publishes "A Mathematical Theory of Communication"
- **1949**: Leon Kraft proves inequality for prefix codes
- **1952**: David Huffman, as a graduate student at MIT, invents optimal prefix coding algorithm
- **1956**: Brockway McMillan extends Kraft inequality to all uniquely decodable codes
- **1978**: Robert Gallager proves tight redundancy bound for Huffman codes

Homework Assignment 2: Source Coding Theory and Huffman Coding

Note: Questions 2 and 3 are comprehensive problems. Question 3 is a challenge problem that explores block coding in depth.

1. **Kraft-McMillan Applications (20 points)**
   (a) Prove that for any uniquely decodable code with codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$, we have $\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$
   (b) Given lengths $\{2, 3, 3, 3, 4, 4\}$, verify if a binary prefix code exists
   (c) If a code exists, construct it using the canonical method
2. **Huffman Code Construction (30 points)**
   (a) For source with probabilities $\{0.35, 0.2, 0.15, 0.1, 0.1, 0.05, 0.05\}$:
   - Construct Huffman tree step by step (show all intermediate steps)

- Assign codewords and calculate expected length
- Compute efficiency and redundancy

(b) Using the lemma from class, prove that in your Huffman code, the two least probable symbols have codewords of equal length

(c) Verify that your code satisfies the Kraft inequality (note: it may not satisfy it with equality if the tree is not complete)

3. **Extended Huffman Coding: Challenge Problem (25 points)**

(a) Consider a binary source with $p(0) = 0.8$, $p(1) = 0.2$

(b) Design Huffman codes for $n = 1, 2, 3$ (code blocks of symbols together)

(c) For each n, calculate:

- Expected length per block L_n
- Expected length per symbol L_n/n
- Efficiency η_n

(d) Plot L_n/n vs n (for $n = 1, 2, 3$) and compare to the entropy limit

(e) Explain mathematically why efficiency improves with n

4. **Optimality Analysis (15 points)**

(a) For the source in question 2, design a Shannon code ($\ell_i = \lceil -\log_2 p_i \rceil$)

(b) Compare the Huffman code with the Shannon code:

- Expected lengths
- Efficiencies
- Redundancies

(c) Under what conditions (what type of probability distributions) does Huffman coding achieve exactly $L = H$?

5. **Practical Considerations (10 points)**

(a) Explain why block Huffman coding with large n is impractical for real applications

(b) What alternative approaches exist for approaching the entropy limit without exponential complexity?

(c) How does arithmetic coding (to be covered in the next lecture) solve the integer constraint problem?

Reading Assignment and References

- **Required Reading**:
  - Cover & Thomas, *Elements of Information Theory*, Chapter 5: Data Compression
  - Shannon, C. E. (1948). "A Mathematical Theory of Communication"
- **Advanced References**:
  - Huffman, D. A. (1952). "A Method for the Construction of Minimum-Redundancy Codes"
  - Gallager, R. G. (1978). "Variations on a Theme by Huffman"
- **Online Resources**:
  - Visual Huffman tree generator: `https://www.csfieldguide.org.nz/en/interactives/huffman-tree/`
  - Kraft inequality calculator: `https://planetcalc.com/8157/`